Conversion of graphs to 2D Image Grids

Thursday, 30th May, 2019

Deepti Saravanan

REFERENCE.

Graph Classification with 2D Convolutional Neural Networks by A. J.-P. Tixier, G. Nikolentzos, P. Meladianos, and M. Vazirgiannis.

Context

- Problem Statement
- Limitations of Graph Kernels
- Methodology
- Overcoming the 3 limitations
- Proposed Method - 3 steps
- 2D CNN

Problem Statement

To represent graphs as multi-channel image-like structures that can be classified into the corresponding categories by the vanilla 2D CNNs.

Limitations of Graph Kernels

1. **High Time Complexity** - Computing the similarity between every two graphs in the training set (say of size N),which amounts to N(N−1)/2 operations. The cost of training therefore increases much more rapidly than the size of the dataset. Also, finding the support vectors is $O(N^2)$ when the C parameter of the SVM is small and $O(N^3)$ when it gets large.
2. **Disjoint feature and rule learning** - The computation of the similarity matrix and the learning of the classification rules are two independent steps. Thus, the features are fixed and not optimized for the task.
3. **Graph comparison is based on small independent substructures** - Graph kernels focus on local properties of graphs, ignoring their global structure.

Methodology

3 steps:

1. Graph node embedding.
2. Embedding space compression.
3. Repeated extraction of 2D slices from the compressed space and computation of a 2D histogram for each slice.

The image representation is finally produced by stacking together the 2D histograms.

(Each histogram as a channel. Image size is same irrespective of the size of the graphs).

Overcoming the 3 limitations

Solution1 - By converting all graphs in a given dataset to representations of the same dimensionality, and by using a classical 2D CNN architecture for processing those graph representations, this method offers constant time complexity at the instance level, and linear time complexity at the dataset level.

Solution2 - In a 2D CNN classifier, features are learned directly from the raw data during training to optimize performance.

Solution3 - Involves state-of-the-art graph node embedding techniques that capture both local and global properties of graphs.

Proposed Method - 3 steps

1. **Graph Node Embeddings** - Local correlation exists - the Euclidean distance between two nodes is inversely proportional to their similarity. Obeys the Spatial Dependency property as required by the CNNs. (*Spatial Dependence* - Parameters influenced by the neighbourhood property).
2. **Alignment and Compression with PCA** (Principal Component Analysis) - Node embedding techniques are stochastic. Thus, the dimensions tend to change from run to run. Thus, PCA is used to retain the first d<<D (D is the dimensionality of the original node embedding space) principal components to ensure that the embeddings of all the graphs in the dataset are comparable. Also, PCA is used for compression that reduces the shape of the tensors fed to the CNN and thus the complexity.

3 steps (contd.)

3. **Computing and stacking 2D histograms** - Repeatedly extract 2D slices from the d-dimensional PCA node embedding space, and turn those planes into regular grids by discretizing them into a finite, fixed number of equally-sized bins, where the value associated with each bin is the count of the number of nodes falling into that bin.

That is, the graph is represented as a stack of d/2 2D histograms of its compressed node embeddings where ‘d’ is the embedding dimensionality of subgraph features.

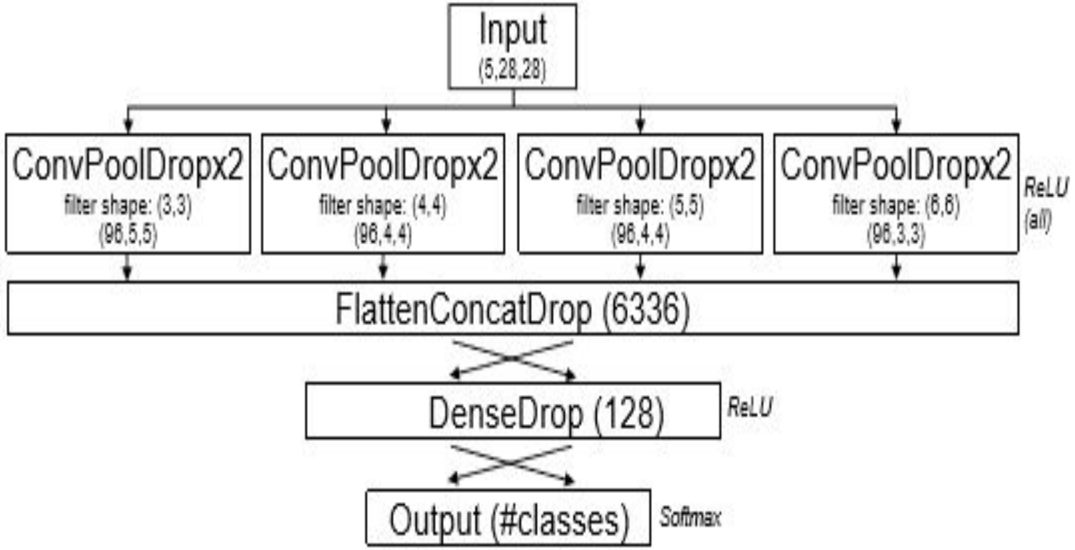
The first histogram is computed from the first two dimensions, the second from the next two dimensions and so on.

The bins can be viewed as pixels and the 2D slices of the embedding space as channels (d/2 in number). The entries of the vector are the counts of the nodes falling into that bin in the corresponding 2D slice of the embedding space.

2D CNN Architecture

- 4 Convolutional pooling layers in parallel with region sizes of 3,4,5 and 6.
- Two fully connected layers.
- Dropout for regularization at every hidden layer.
- ReLU functions for activations(softmax for the last layer).
- Convolutional Pooling Block - 64 filters at 1st level, the number of filters increased to 96 in the subsequent convolutional layer to compensate for the loss of resolution as the signal is halved through the (2,2) max pooling layer.

Output - Probability distributions over classes.


Input
(5,28,28)
ConvPoolDropx2
filter shape: (3,3)
(96,5,5)
ConvPoolDropx2
filter shape: (4,4)
(96,4,4)
ConvPoolDropx2
filter shape: (5,5)
(96,4,4)
ConvPoolDropx2
filter shape: (6,6)
(96,3,3)
ReLU
(all)
FlattenConcatDrop (6336)
DenseDrop (128)
ReLU
Output (#classes)
Softmax

THANK YOU